Aufgabe 1: Programmieren mit Smalltalk (26 Punkte = 6 + 4 + 8 + 8 P.)

a) Wie kann man das aus anderen Sprachen bekannte `If-then-else`-Konstrukt in SMALLTALK realisieren? Schreiben Sie bitte die entsprechenden Methodendefinitionen auf.

Lösung:

für true:

```
ifTrue: wahrBlock else: falschBlock
  ^ wahrBlock value
```

für false:

```
ifTrue: wahrBlock else: falschBlock
  ^ falschBlock value
```

b) Auf dem Transcript soll die Folge der Ziffern 5, 3, 1 ausgegeben werden. Geben Sie die entsprechenden Smalltalk-Befehle an.

Lösung:

```
5 to: 1 by: -2 do: [ :i | Transcript show: i printString]
```

oder

```
#(5 3 1) do: [ :i | Transcript show: i printString]
```

c) Wie kann man die aus anderen Sprachen bekannte `while`-Schleife in SMALLTALK realisieren? Schreiben Sie bitte die entsprechenden Methodendefinitionen auf.

Lösung:

für true:

```
whileTrue: aBlock
    self value
       ifTrue: [
       aBlock value.
       self whileTrue: aBlock].
    ^ nil
```

für false:

```
whileFalse: aBlock
    self value
       ifFalse: [
       aBlock value.
       self whileFalse: aBlock].
    ^ nil
```

d) Der Ausdruck

```
freund einladen
```

besagt, daß das Objekt, mit dem der Besitzer der Variable `freund` in gleichnamiger Beziehung steht, die Nachricht `einladen` erhalten soll.
Wie sieht der entsprechende Ausdruck aus, wenn mehrere Freunde eingeladen werden sollen?
Wie sieht der entsprechende Ausdruck aus, wenn nur enge Freunde eingeladen werden sollen?
(Hinweis: Sie dürfen die Methode `select` als bekannt voraussetzen.)

Lösung:

Einladung von mehreren Freunden:

```
freunde do: [ :freund | freund einladen]
```

Einladung von engen Freunden:

```
(freunde select: [ :freund | freund eng = true])
   do: [ : freund | einladen]
```

Aufgabe 2: Grundbegriffe (12 Punkte = 6 + 4 + 2 Punkte)

a) Ordnen Sie der nachfolgenden Sequenz von Ist-ein-Sätzen die jeweilige Form der Abstraktion zu:

1. Pepe ist ein Pferd.
2. Pferd ist ein Säugetier.
3. Säugetier ist ein Wirbeltier.
4. Wirbeltier ist ein Stamm.
5. Stamm ist ein Taxon.
6. Pferd ist eine Spezies.
7. Spezies ist ein Taxon.

Bestimmen Sie die längste Folge der daraus ableitbaren Ist-ein-Sätze, bei der die Klassifikation nicht wechselt.

Lösung:

1. Pepe ist ein Pferd. **Form der Abstraktion = Klassifikation**
2. Pferd ist ein Säugetier. **Form der Abstraktion = Generalisierung**
3. Säugetier ist ein Wirbeltier. **Form der Abstraktion = Generalisierung**
4. Wirbeltier ist ein Stamm. **Form der Abstraktion = Klassifikation**
5. Stamm ist ein Taxon. **Form der Abstraktion = Klassifikation**
6. Pferd ist eine Spezies. **Form der Abstraktion = Klassifikation**
7. Spezies ist ein Taxon. **Form der Abstraktion = Klassifikation**

Längster ableitbarer Satz: $\textit{Pepe} \in \textit{Pferd} \in \textit{Spezies} \in \textit{Taxon}$

b) Was ist der Unterschied zwischen Spezialisierung und Instanziierung?

Lösung:
Spezialisierung entspricht der Teilmengenbeziehung, Instanziierung entspricht der Elementbeziehung.
Bei der Spezialisierung wird aus einer Klasse eine andere gebildet, die spezieller ist.
Bei der Instanziierung wird aus einer Klasse ein Objekt gebildet.
Bei der Spezialisierung wird der Wertebereich eingeschränkt.
Bei der Instanziierung wird (ggf. über eine Initialisierung) einer Instanzvariable eines Objekts *ein Element* aus dem Wertebereich zugewiesen.

c) Darf man bei der Spezialisierung Instanzvariablen und Methoden wegnehmen?

Lösung:

Nein, das ist nicht erlaubt.

Aufgabe 3: Spezialisierung/Generalisierung (10 Punkte = 5 + 2 + 3 P.)

In einem SMALLTALK-Projekt ist die Klasse `Quadrat` schon vorhanden. Sie hat die benannte Instanzvariable `laenge` und die Instanzmethoden

```
flaeche
  ^ laenge * laenge

umfang
  ^ laenge * 4
```

Nun möchte man ein zweite Klasse `Rechteck` definieren und dabei ausnutzen, daß man so eine ähnliche Klasse, nämlich `Quadrat`, schon hat. Aus `Quadrat` übernehmen läßt sich nämlich die Instanzvariable `laenge`.

a) Wie muss die Klasse Rechteck dann aussehen?

Lösung:

Klasse: Rechteck

beerbte Klasse: Quadrat

benannte Instanzvariable: breite

Instanzmethoden:

```
flaeche
  ^ laenge * breite

umfang
  ^ (laenge + breite) * 2
```

b) Konnten die Methoden `flaeche` und `umfang` der Klasse `Rechteck` von der Klasse `Quadrat` geerbt werden? Wenn ja, warum? Wenn nein, was haben Sie stattdessen gemacht?

Lösung:
Nein, die Methoden müssen neu definiert (*überschrieben*) werden.

c) Hat man jetzt gleichzeitig mit der Vererbung auch eine Spezialisierungs- bzw. Generalisierungsbeziehung geschaffen? Begründen Sie bitte Ihre Antwort kurz!

Lösung:

Nein, denn:
Die Menge der Quadrate enthält die Menge der Rechtecke nicht, was ja eine charakteristische Begleiterscheinung der Generalisierung gewesen wäre.

Aufgabe 4: Typsystem (10 Punkte = 4 + 2 + 4 Punkte)

a) Warum typisiert man Variablen und andere Programmelemente? Nennen Sie bitte vier Gründe! Welcher Grund ist für die objektorientierte Programmierung am wichtigsten?

Lösung:

1. Typisierung regelt das Speicher-Layout.
2. Typisierung erlaubt die effizientere Ausführung eines Programms.
3. Typisierung erhöht die Lesbarkeit eines Programms.
4. Typisierung ermöglicht das automatische Finden von logischen Fehlern (semantischen Fehlern) in einem Programm.

b) Wenn man eines der heute üblichen Typsysteme verwendet, gibt es noch einen fünften Grund, warum man das tun sollte. Welcher ist das?

Lösung:

Ein fünfter Grund zur Verwendung eines der heute üblichen Typsysteme ist die dadurch entstehende Modularisierung von Programmen, nämlich wenn ein Typ zugleich eine Schnittstelle oder ein Interface ausdrückt.

c) Was muß ein Typsystem im Programm überprüfen, um Freiheit von semantischen Fehlern garantieren zu können?

Lösung:

Der einzige Weg, eine mit der Typisierung einer Variable ausgedrückte Invariante zu verletzen, also Typinkorrektheit herzustellen, ist per Wertzuweisung an die Variable. Ein Typsystem muß also lediglich alle Wertzuweisungen in einem Programm überprüfen, um Freiheit von semantischen Fehlern zu garantieren. Dazu zählen allerdings auch die impliziten Zuweisungen bei Methodenaufrufen, die, auch wegen des dynamischen Bindens, nicht immer alle offensichtlich sind.

Aufgabe 5: Polymorphismus (24 Punkte = 3 + 9 + 3 + 3 + 6 P.)

a) Typen beschränken die Wertebereiche von Variablen und Methoden. In wieweit lockert die Inklusionspolymorphie diese Beschränkung auf?

Lösung:

Inklusionspolymorphie lockert diese Beschränkung insofern auf, als dadurch Wertebereiche von Typen um die von Subtypen erweitert werden können, selbst wenn diese Subtypen zum Zeitpunkt der Typdefinition noch gar nicht bekannt waren.

b) Was versteht der Kurs 1814 unter parametrischem Polymorphismus?
Beantworten Sie dazu die folgenden Fragen:

1. Was versteht man unter einer parametrischen Typdefinition?

2. Was ist die Instanziierung des parametrischen Typs?

3. Was ist die Idee des parametrischen Polymorphismus?

Lösung:

1. Eine parametrische Typdefinition unterscheidet sich von einer normalen dadurch, daß in der Typdefinition verwendete, andere Typen nicht genannt (referenziert) werden müssen, sondern durch Platzhalter, die Typparameter, vertreten werden können.
   Diese Platzhalter sind Variablen, deren Wert implizit (also ohne entsprechende Deklaration) auf Typen beschränkt ist; man nennt sie auch Typvariablen.

2. Die Typvariablen werden erst bei der Verwendung eines parametrisierten Typs in der Deklaration eines anderen Programmelements mit einem Wert, also einem Typ, belegt. Man spricht bei dieser Wertzuweisung an eine Typvariable von einer Instanziierung des parametrischen Typs; erst bei ihr entsteht ein konkreter Wertebereich, der dann dem deklarierten Programmelement zugeordnet wird.

3. Die Idee des parametrischen Polymorphismus ist, aus einer Typdefinition durch Parametrisierung viele zu machen. Eine parametrische Typdefinition steht also nicht für einen Typ, sondern für (theoretisch) beliebig viele; sie erlaubt es gewissermaßen, Typen nach Bedarf zu generieren.

c) Nennen Sie einen Standardanwendungsfall für den einfachen parametrischen Polymorphismus und beschreiben Sie diesen Anwendungsfall kurz.

Lösung:

Collections sind ein Standardanwendungsfall. In der Collection sollen Objekte eines bestimmten Typs gespeichert werden. Oft möchte man sich aber nicht auf einen bestimmten Typ für diese Objekte festlegen, sondern die Collection, z.B. ein Dictionary, allgemein definieren und erst später angeben, für welche Typen von Objekten sie verwendet werden soll.

d) Warum ist in bestimmten Situationen der einfache parametrische Polymorphismus dem Inklusionspolymorphismus vorzuziehen?

Lösung:

Der einfache parametrische Polymorhismus behebt eine Sicherheitslücke in der statischen Typprüfung, die bei der Verwendung von Inklusionspolymorphie auftreten kann.
Begründung:
Zur Erläuterung wählen wir die Sprache STRONGTALK. Dort sind alle Typen Subtypen von `Object`. Also könnte man die Collection für Objekte vom Typ `Object` definieren und damit jedes beliebige Objekt in der Collection speichern. Sei `Person` der Typ solch eines Objektes. Damit man die speziellen Eigenschaften von `Person` ausnutzen kann, muss man einen sogenannten Downcast durchführen, d.h. eine Umwandlung in den Typ `Person`. Die Zulässigkeit der Typumwandlung ist aber davon abhängig, was wirklich in der Collection drinsteckt, und das kann der Compiler nicht (oder nur sehr aufwendig) feststellen. Es tritt also eine Sicherheitslücke in der statischen Typprüfung auf.

e) Welche Unzulänglichkeiten des einfachen parametrischen Polymorphismus sind zu beachten?

Lösung:

Unzulänglichkeit 1:
Ohne Inklusionspolymorphie ist es nicht möglich, in einer Collection mit Elementtyp XYZ auch Objekte eines Subtyps von XYZ zu speichern (heterogene Collections).

Unzulänglichkeit 2:
Die erhöhte Typsicherheit bei der Verwendung von parametrisch definierten Typen wird mit einer geringeren Typsicherheit innerhalb der Typdefinition (bzw. Klassendefinition) selbst erkauft. Gegeben ist z.B. der Collection-Typ MyCollection, dessen Werte solche Collections sein sollen, deren Elemente sortiert und summiert werden können. Dieser Typ sei ein Subtyp von Collection und verfüge weiterhin über entsprechende Methoden sortieren und summieren. Nun kann aber die Definition des parametrischen Typs MyCollection nicht wissen, wie sie hinterher verwendet wird, und wenn eine Addition durchgeführt werden soll, ist sie darauf angewiesen, daß sie nur mit Typen von addierbaren Objekten **instanziiert** wird. Wenn das nicht geschieht, gibt es einen Typfehler.

Aufgabe 6: Java (8 Punkte)

Gegeben ist der folgende Programmausschnitt in Java:

```
class Hund extends Tier {}
Tier[] tiere;
Hund[] hunde;
```

Es werden die folgenden beiden Zuweisungen gemacht:

```
tiere = hunde;
tiere[1] = new Tier();
```

Sind diese Zuweisungen in Java erlaubt? Begründen Sie Ihre Antwort.

Lösung:

Die Zuweisung `tiere = hunde` kann man in Java durchführen.

Die anschließende Zuweisung `tiere[1] = new Tier()` führt dann in JAVA allerdings prompt zu einem Laufzeitfehler (zu einer sog. Array store exception),
denn:
`tiere` ist ja lediglich ein Alias auf ein Array mit Hunden, so daß die Zuweisung ein Tier anstelle eines Hundes an Arrayposition 1 setzt und das Array `hunde`, das ja per Deklaration nur Hunde zu enthalten verspricht, damit nicht mehr typkorrekt ist.
Würde man also diese Zuweisung `tiere[1] = new Tier()` zulassen, dann würde in der Folge die scheinbar korrekte Zuweisung `Hund hund = hunde[1]`, bei der `hund` ein Tier zugewiesen wird, die Typinvariante von `hund` verletzen.

Aufgabe 7: Schnittstellen (15 Punkte = 2 + 5 + 3 + 5 P.)

a) Was ist die öffentliche Schnittstelle einer Klasse in Java?

Lösung:

Die öffentliche Schnittstelle einer Klasse in JAVA ist die Menge ihrer Eigenschaften (Instanzvariablen und -methoden), die als public deklariert sind.

b) Was ist ein Java-Interface?

Lösung:

Es gibt in JAVA die Möglichkeit, die öffentliche Schnittstelle einer Klasse als eigenständiges Konstrukt zu deklarieren, das zunächst von dem der Klasse unabhängig ist, das aber genauso wie eine Klasse einen Typ definiert. Es geschieht dies mit Hilfe des Schlüsselwortes interface:

```
interface <Interfacename> {
   <Rueckgabetyp 1> <methode>(<Parametertyp 1> <parameter 1>, ...);
    ...
}
```

Interfaces liefern definitionsgemäß weder Implementierungen noch Objekte (Instanzen).

c) Wie kann man Java-Interfaces in einem Programm sinnvoll einsetzen?

Lösung:

Möglichkeit 1: Interfaces bieten klientenspezifische Sichten.
Möglichkeit 2: Interfaces gestatten Austauschbarkeit einer Klasse gegen andere, ohne gleichzeitig zu erben.
Möglichkeit 3: Man kann sie als Tagging oder Marker Interfaces verwenden (Beispiel: Serializable).

d) Die ´Mehrfachimplementierung´ von Interfaces wird häufig als Ersatz für die in Java fehlende Möglichkeit der Mehrfachvererbung angepriesen. Nehmen Sie bitte dazu Stellung.

Lösung:

Eine Klasse in JAVA kann nur direkte Subklasse genau einer anderen Klasse sein, dafür aber mehrere Interfaces gleichzeitig implementieren. Diese mögliche ´Mehrfachimplementierung´ von Interfaces wird häufig als Ersatz für die in JAVA fehlende Möglichkeit der Mehrfachvererbung angepriesen; das aber ist Unsinn, denn bei der Implementierung eines Interfaces wird nichts vererbt. Vielmehr hat man es mit einer Art Mehrfach-Subtyping zu tun.

Aufgabe 8: C#, C++, EIFFEL (12 Punkte = 4 + 4 + 4 P.)

a) Was ist das Besondere an der Typhierarchie von C#?

Lösung:

In C# sind genau wie in JAVA alle Variablen typisiert.
Anders als in JAVA wird dabei jedoch zunächst nicht zwischen Wert- (primitiven) und Referenztypen unterschieden: Alle Typen, auch die primitiven, gelten als von Object (genauer: System.Object) abgeleitet.

b) Welches Konzept von C++ hat einen direkten Bezug zur objektorientierten Programmierung?
Was kann man damit erreichen?

Lösung:

Ein interessantes Konzept von C++, das einen direkten Bezug zur objektorientierten Programmierung hat, ist das Friends-Konzept. In der Praxis kommt es häufig vor, daß ein bestimmtes Teilproblem nicht von einer Klasse allein, sondern nur durch das Zusammenspiel mehrerer Klassen gelöst werden kann. Während diese Klassen untereinander eng kooperieren müssen und deswegen (relativ) intime Kenntnis voneinander benötigen (will sagen, auf Elemente zugreifen können müssen, die anderen Klassen verborgen bleiben sollten), gilt das für andere Klassen nicht unbedingt. Die Schnittstelle solcher kooperierenden Klassen sollte also nicht absolut, sondern relativ zu anderen Klassen definierbar sein.

c) Was sind die vier prominentesten Eigenschaften des Typsystems von EIFFEL?

Lösung:

1. Mehrfachvererbung
2. beschränkter parametrischer Polymorphismus
3. das Unterdrücken von Instanzvariablen und Methoden in Subklassen (Löschen von Methoden)
4. kovariante Redefinition, unterstützt durch sog. verankerte Typen

Aufgabe 9: Probleme der OOP (10 Punkte = 4 + 6 Punkte)

a) Beschreiben Sie bitte kurz das Fragile-base-class-Problem.

Lösung:

Zugrunde liegt den zahlreichen Varianten des Fragile-base-class-Problem immer dasselbe: Zwischen einer Klasse und ihren Subklassen bestehen durch die Vererbung von Eigenschaften starke Abhängigkeiten, die - wenn überhaupt - nur unvollständig dokumentiert sind.

b) Nennen Sie die weiteren im Kurs behandelten Probleme der objektorientierten Programmierung.

Lösung:

1. Problem der Substituierbarkeit
2. Problem der schlechten Tracebarkeit
3. Problem der eindimensionalen Strukturierung
4. Problem der mangelnden Kapselung
5. Problem der mangelnden hierarchischen (De-)Komposition und mangelnden Skalierbarkeit
6. Problem der mangelnden Eignung

Aufgabe 10: Objektorientierter Stil (8 Punkte = 2 + 4 + 2 P.)

Das Gesetz Demeters lautet: *Sprich nicht mit Fremdem!*

a) Was besagt dieses Gesetz im Zusammenhang mit objektorientierter Programmierung?

Lösung:

Nachrichten dürfen nur an Objekte versendet werden, die der Sender selbst kennt oder erzeugt.

b) Wann kennt ein Objekt ein anderes?

Lösung:

Ein Objekt kennt ein anderes, wenn es in direkter Beziehung dazu steht, wenn es also auf eine Variable direkt (also ohne den Umweg über ein anderes Objekt) Zugriff hat, die das andere Objekt benennt.
Es kennt das andere Objekt dauerhaft (oder zumindest für eine längere Dauer), wenn es sich bei der Variable um eine Instanzvariable handelt, und temporär, wenn es sich bei der Variable um den formalen Parameter einer Methode handelt (wobei die Dauer des Kennens dann auf die Dauer der Abarbeitung der Methode beschränkt ist, es sei denn, der formale Parameter wird einer Instanzvariable zugewiesen).

c) Nennen Sie bitte ein Beispiel dafür, daß man in der objektorientierten Programmierung eine Ausnahme vom Gesetz Demeters machen muß.

Lösung:

Bei der Verwendung von Collections als Zwischenobjekte muß man immer eine Ausnahme von Demeters Gesetz machen.